React Ref best practices

useRef is often used as a reference to some DOM element, when react API is not enough.

useRef can be used also if you want to have a variable, that will keep some information between rendering and you don’t want to call page rerendering after changing this variable (unlike useState).

useRef could be used only on native DOM elements (HTML tags).

if you want to pass ref as a prop to the child component, you should wrap your child component into forwardRef HOC.

React Context

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

In a typical React application, data is passed top-down (parent to child) via props, but such usage can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language

1. You can and should have multiple context providers for different state that needs to be disseminated. For example, if we wanted to disseminate the current user and the color theme in the Help Queue, those should be in two separate contexts.

2. Locate your provider as close as possible to the consumer components. This is a general best practice that may not be possible in some cases. For example, when your consumer components are nested in your component tree and they are on different branches, it often will not be possible to locate your context provider that close to any consumer components.

3. Make sure that only the components that need the context data, have it. You shouldn't make a component a consumer if it doesn't absolutely need to. Keep in mind that anytime the context provider's value changes, all components that consume that context value will re-render.

4. Use context judiciously. Context should be the last tool you reach for, before prop drilling, lifting state up, and composition. Using context in components makes them harder to reuse. How so? Because any consumer component needs to be located downstream of a provider component. So, before you use context, ask yourself if the data that you need to disseminate is really needed on a global or wide scale. Also, before reaching for context, first try using props and composing your components to make passing props easier.